A Flexible Monitoring Infrastructure for the Simulation Requests

V Spinoso[1], **M Missiato**[1]
[1]INFN, Bari, Italy

E-mail: `vincenzo.spinoso@ba.infn.it`

Abstract. Running and monitoring simulations usually involves several different aspects of the entire workflow: the configuration of the job, the site issues, the software deployment at the site, the file catalogue, the transfers of the simulated data. In addition, the final product of the simulation is often the result of several sequential steps. This project tries a different approach to monitoring the simulation requests. All the necessary data are collected from the central services which lead the submission of the requests and the data management, and stored by a backend into a NoSQL-based data cache; those data can be queried through a Web Service interface, which returns JSON responses, and allows users, sites, physics groups to easily create their own web frontend, aggregating only the needed information. As an example, it will be shown how it is possible to monitor the CMS services (ReqMgr, DAS/DBS, PhEDEx) using a central backend and multiple customized cross-language frontends.

1. Introduction

Event generation, simulation and reconstruction belong to the daily activities of the CMS experiment. Very complex procedures are generally used to generate a given sample: the users have to define separate subsequent steps, which are submitted to the computing infrastructure, and translated to hundreds of thousands of jobs per day, creating all the needed intermediate samples and, of course, the final sample.

In particular, the preparation of the simulation requests and their execution have recently taken the challenge of producing more than 10B events per year. The key of this success is *automation*; indeed PREP and the new McM [1] have had a crucial role in helping the users in the *preparation* of the simulation requests, providing them with new tools and concepts like *campaigns* and *chains of requests*. On the other hand, the new well-scaling production infrastructure [2] built on Request Manager [3], WMAgent and glideinWMS [4], have permitted the centralized submission of the requests, allowing an unprecedented usage of the computing resources.

2. Monitoring the simulation requests

The activities just described can be monitored through specific monitoring tools. For instance, McM itself mantains the status of a given request, as it interacts with DAS [5] and Request Manager to aggregate some information about the execution of the running requests and the generated output; besides, the production infrastructure is monitored by a dedicated monitoring system called WMStats, able to show the details and the status of each request submitted to the sites.

Unfortunately the monitoring solutions just mentioned are not oriented to the final user: it is not easy to follow the evolution of the generation of a given chain of requests using a single and easy to use monitoring system; moreover, often the technical details published by "dedicated" monitoring interfaces are misleading. For instance, McM doesn't show the evolution of the requests after the submission and until the generation is completed; for that reason, it is necessary to access WMStats, which shows many technical details useful for the daily job of central operations, but difficult to understand to the average CMS user. Also, a separate tool like DAS must be used to access the details of the output *datasets*, and the PhEDEx web pages are needed to subscribe the transfer of a specific sample to a specific CMS Tier2. Unfortunately the generic CMS user is not (and is not supposed to be) skilled enough to interact with all of them.

3. The Production Aggregated Monitoring

All the services just mentioned publish *web services*: this means that each application can be queried remotely. This is the idea on which PAM is built.

PAM, acronym of *Production Aggregated Monitoring*, represents an abstraction layer on top of CMS central services. PAM is designed to be very *user friendly*: it enables the generic CMS user to access an aggregated view of the simulation requests in a very simple way; no complicated API has to be used, all that must be provided to PAM is a simple information like the ID of a specific request, for which PAM returns a JSON structure with the details. The data format used by PAM to store and return the information is JSON, because JSON is supported by all the major programming languages, and it usually allows to easily translate a given JSON data structure into the chosen programming language using a single call to a JSON library. On the other hand, PAM is not a *web monitoring interface*; instead, it has to be intended as a *data source* where users can get their information, using, for instance, a cronjob, and arrange them as they need, creating web pages or sending the output of the cronjob directly to their email address. Using PAM as a *swiss knife* to get the details of a request, the direct interaction with the CMS central services can be avoided.

In principle the PAM user can be also a skilled web developer, which focuses the development more on the choice of the data, on the way in which they are organized, on the layout of the page, getting rid of all the details about the interaction with the CMS services.

PAM is made of two logical components: the Backend and the Frontend.

3.1. The Backend

The Backend queries the CMS services about requests and datasets; it collects the results into JSON structures and stores them in a local cache; if the same call is made to the Backend afterwards, it can return the cached results instead of querying the services once again, depending on the age of the data themselves in the cache; this protects the services from hammering (queries made with high frequency), and provides a fast response when the result is already available in the cache.

The format used by the Backend to store the data about requests and datasets is a mixture of flat files and CouchDB. The migration to full CouchDB is currently ongoing. CouchDB represents a good compromise between the simplicity of the flat-file format and the power of a DBMS. The first version of PAM was based on Oracle, but that solution was abandoned soon for the following reasons:

- Oracle, just like any other SQL-based DBMS, requires the creation of a database, with a corresponding *schema*. During the last two years, CMS services have been revolutionized, and several changes were done on a monthly basis; so the adoption of a DBMS schema has been proved not to be *flexible and sustainable* enough;

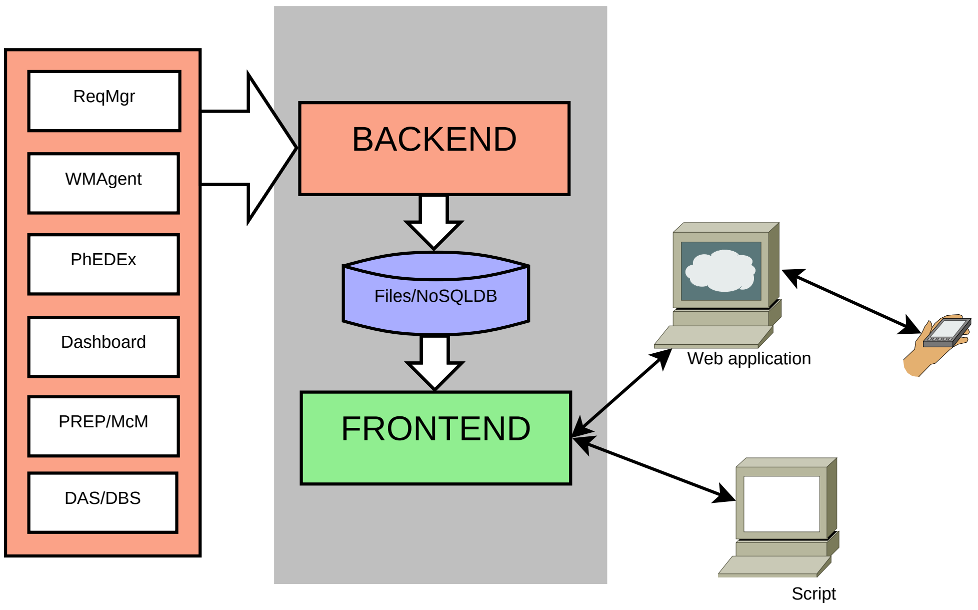

Figure 1. PAM components and interfaces: overview

- only a standard Python 2.6 environment and a portable library (couchdbkit) are needed to interact with CouchDB;
- data organized in JSON format are just ”ready” to be stored in the CouchDB, as CouchDB is *document oriented*;
- using a DBMS makes sense if typical DBMS features are needed, like automatic backup, high availability, partitioning. PAM does not need any particular DBMS features, as it acts like a simple cache of data: even if the whole content of the cache is lost, all the data can be just reread from the sources.

Only the most useful information are stored in the PAM cache. As soon as a new parameter must be added into the cached details of a request, it can be quickly added into the JSON infrastructure, without impacting on the data already stored, and without the need of modifying the format used to store them. Intervention on the code is needed only if major changes happen on the infrastructure behind the Backend.

3.2. The Frontend
The Frontend connects the user with the Backend. It publishes a *REST web service*, serving few very general calls:

- it is possible to get the IDs of all the requests having a given *status* or *type*; for instance, it’s possible to get the IDs of all the *reprocessing* requests that are *running*;
- additionally, the explicit ID can be specified as input: in that case, the call will return all the requests having the same ID. In particular, the request having a given *PREPID* in the McM context, can be identified by multiple *request names* in the Request Manager; those requests are called *clones*;
- the most useful search parameter that is available is the *primary dataset*, which is the part of a dataset name that expresses the physical content of the dataset;
- all the searches above support *regular expressions* to provide maximum flexibility.

In order to use PAM, it is necessary to have

- a CERN account on LXPLUS, and a CMS affiliation;
- a regular personal digital certificate, provided by a regular Certification Authority like CERN.

Once logged into LXPLUS, it is already possible to use PAM through the following service:

`http://pam.cern.ch/pamws.py/Resource?parameters=values`

using Python scripts and cronjobs.

With the help of an SSH tunnel, it is possible to provide access to a host which is external to the CERN network; in that way, a PAM client can also run outside CERN. A real use case corresponds to the Higgs and SUSY local groups of the CMS Tier-2 at Bari, which are now able to monitor their requests using a simple web page refreshed by a cronjob, running on a server of the Tier2. The web page provides information about the requests currently submitted to the computing infrastructure; for each request, it is provided in particular:

- **type** and **status** of the request: the terminology used by the Request Manager and McM is translated into a more friendly indication. For instance the "MonteCarloFromGEN" type of the Request Manager is represented as "step1", and the "running-closed" status is represented as "running"; another example is the case of requests which are submitted from McM but are not yet really running, which are grouped under the same voice "Submitted to Computing" instead of using separate statuses inherited from Request Manager ('assignment-approved', 'assigned', 'acquired');
- **last update**: it corresponds to the timestamp of the last update in the PAM cache;
- **primary dataset**: this really helps the final users in identifying the sample they are interested in;
- **output dataset**.

All the information provided are linked to the CMS central pages; for instance, it's possible to click on the output dataset to load the DAS page about that dataset. For proper reference, the web page can be found at the following URL:

4. Chained requests

The generation of a sample needs the generation of intermediate samples. Each sample is generated starting from a request, which is a set of settings, directives, parameters used by the computing infrastructure to produce a sample (intermediate of final). Two requests are *chained* if the output of the former is the input of the latter.

PAM uses a special graph-based algorithm, based on the *Breadth-first search* algorithm, which is general, exhaustive and fast, in order to match requests which are connected each other through a given dataset. In particular, starting from a given request or dataset, it's possible to get a list of all the requests that are chained to that particular request.

Concerning chained requests, the most important consideration is that the users can check how the requests are *really connected*. Let's focus on a use case taken from real daily central operation.

After the resubmission of a request, it's possible that the original request is left running: this can be due to several reasons (status update fails, human mistakes...). In that case, PAM returns a list where two requests "point" to the same output dataset, no matter what is foreseen by McM in the original request; WMStats, on the other hand, doesn't even support the concept of chained request.

An important aspect of the chained request iconcept s that, despite most of the operations are automated, each step of the chain, namely each request, needs to be *defined* and *validated* manually. The final user can use PAM to monitor the whole chain: if a given request remains stuck for days for some reasons, he can alert central operations; hence, PAM can contribute in *reducing latencies*.

5. Conclusions

The Production Aggregated Monitoring is a framework created to help the final users monitor their Monte Carlo requests. In order to allow the user to focus on the *status* of the requests, which is the information that really matters until the completion of the production, PAM provides a very simplified view of the request itself. It's possible to:

- follow the evolution of a given request, without asking the production teams;
- get a new overview of the whole chain of subsequent requests;
- detect unwanted and unpredictable issues happening in the computing infrastructure.

The abstraction performed by PAM allows the developer to focus on a very complex arrangement of the data, without taking care (and even learning) the details of the interaction with the CMS services. The architecture is very simple and flexible, and goes towards the direction of the *sustainability* of the project.

The concept of “final user” can be extended, as shown, to a physics group, to a Tier2 community or, even, to the CMS entire collaboration.

Acknowledgements

This work was supported by the MIUR (Italian Ministry for Education, University and Research) in the “PON Ricerca e Competitivita' 2007-2013 - Azione I interventi di rafforzamento strutturale - PONa3_00052, Avviso 254/Ric - ReCaS”

References

[1] Vlimant JR, “MCM: The Evolution of PREP. The CMS tool for Monte Carlo Request Management”, Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP13), Amsterdam, The Netherlands, October 2013

[2] Cinquilli M et al 2012, “The CMS workload management system”, J. Phys.: Conf. Ser. 396 032113, doi:10.1088/1742-6596/396/3/032113

[3] Fajardo E et al 2012 “A new era for central processing and production in CMS” J. Phys.: Conf. Ser. 396 042018, doi:10.1088/1742-6596/396/4/042018

[4] Gutsche O et al. 2013 “Evolution of the pilot infrastructure of CMS: towards a single glideinWMS pool”, Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP13), Amsterdam, The Netherlands, 14 Oct - 18 Oct 2013

[5] Kuznetsov V et al., “The CMS data aggregation system”, ICCS, 2010, doi:10.1016/j.procs.2010.04.172